ICFINT Webinar: XML Concepts Tech Training Video

March 22, 2023

Presenter: Brendan Martin, AFCARS Technical Administrator

Brendan Martin: Brendan Martin, your AFCARS Technical Administrator, and today I'd like to introduce a technical subject. However, my intended audience is mostly non-technical people such as program administrators and child welfare workers. As you know, AFCARS is upgrading to a new version in which data is submitted in the form of XML. As a former software developer myself, I can appreciate how it can be intimidating to talk to software developers who use a lot of acronyms, and today's acronym is XML. Don't let it intimidate you. It's a very easy to grasp concept, and it's worth grasping because it'll help you understand your business processes. So what does XML mean? XML means extensible markup language. The M is pretty self-evident. It means data is marked up with tags. You're familiar with tags. That means an angle bracket around a term or a description of the data within it. The X is less clear. X means extensible. What does that mean? Extensible means, as you can see I've highlighted here in yellow, is that you can--it's pretty easy to extend a data object or an element. Here you might want to add some more physical characteristics, or you might want to add some other features of the person in this data.

A couple items of terminology that are probably worth knowing if you want to converse cogently about XML. The person, the name, the address, et cetera, these are all called elements. These items in red, which you can kind of glean are descriptive of elements generally, these are called attributes. Technically, you could include the data included in these attributes as elements themselves. For example, you could have--you could have his height as an object, you know, as an element called height. However--and there's no rule against that. However, generally attributes are used--that is, these red descriptors within the opening--within the opening tag of an element, these are generally used when you're describing some kind of object or thing, and a good example of when you would use them is, for example, if you wanted to print data, attributes might be used to describe how something should be printed in bold, et cetera, et cetera.

Now from the previous screen, you could pretty much guess what the data was about because it had tags that described it. This screen is

showing you what current AFCARS data looks like. The bottom of the screen, you can see long records. It's hard to even tell where they end. So my question would be it's pretty illegible. How does one read this data? Anybody have an idea? Well, in fact, the only way to read this data is to write a specific computer program that would parse it and read it and know what piece of data is at what part of what record, how long each record is, et cetera. It is not easily extensible. If you wanted to stick a new data element in there, you'd have to redefine the computer program, and you'd have to make sure that anyone using that computer program is aware of that change, et cetera, et cetera.

XML, on the other hand, as we've already seen, is quite legible. This, you can see, is a person. Now I earlier mentioned HTML because you're familiar with that. HTML is--those tags in HTML always mean the same thing. If you have a B tag, it means bold. If a U tag, it means underline. Is there anything about this XML data that makes this name a name? Is there anything about this that makes it a street, et cetera? What exactly makes this data function as a name or an address, et cetera? And let's take an example, a kind of counter example. If you look at this snippet of XML below it, would you guess that this XML is valid? Anyone in the audience? Is it valid or no? OK. No takers? No one wants to answer the question, but in fact, sure, it is valid. If you and the person you're exchanging data with agree you want to have nonsensical names for a person and his address, you can call these elements whatever you wish.

So how, given that it's a matter of a protocol between two partners exchanging data, how does one agree on what the protocol is and what the data means? The answer to that is using what's called a schema, and the schema defines--it doesn't define the meaning of the data, but it does define what your XML file will contain and how you must build that XML file and what data must be excluded or can be excluded. So hopefully you can see this screen. This is actually an early--one of the earliest ways of making a schema to define data. This strange syntax you see at the top of the XML file is called a DTD, and it used to be you would, you know--you would stick the data--it stands for data type definition, I believe. It used to be that you would simply stick that definition into the file itself. So down here in the bottom half of this screenshot, you can see XML that's describing Bollywood movies. The top part, it's a little more cryptic, but if you look kind of closely, it's pretty intuitive what it means. You have elements, which I've already explained, and for example, here at the bottom, you have an attlist, an attribute list. So a category element can have a type, and

you can see down in the XML there are thriller and romance types. It can have a type, and then within parentheses you can see here--you can probably guess what this means. You have options of these 4 options separated by vertical bars, which means or, and finally, you might even be able to guess what this drama in quotation marks means. That means that would be the default. If you do not specify a category type, it would default to drama. Other elements, other cryptic kind of items here are that, for example, a dvdlibrary has to have--is described as having a movie, and there's a plus sign, and that would mean it has to have one or more movies. You can't submit an empty DTD library, et cetera. The details are not that important.

Oh, one other thing I would like to--a matter of terminology that's probably useful goes along with the idea that XML is pretty intuitive, it's pretty legible, visibly comprehensible. The elements--actually, the technical term for elements within elements is their children. In this example, a movie is a child of dvdlibrary, title is a child of movie. Dvdlibrary is a parent of movie. Is it a parent of title? No, it's an ancestor of title, And is title a child of dvdlibrary? No, it's a descendant. So you have you have two axes, descendant-ancestor. An immediate ancestor is a parent, and an immediate descendant is a child, and children at the same level, they're called siblings. So all of this is plain English that describes it, and in fact, in the kind of computer programing you do to parse or transform XML, you would use these terms. So anyway, this is--again, this is an example of an early form of defining data through a DTD.

Now we're not gonna be using DTDs in AFCARS data. We're gonna be using a different kind of schema, and it's called XML schema. Another point I want to make here, though, highlighted in yellow is it's pretty clear you probably don't want to include the definition of the data in the file itself because many files will share the same definition, and you don't you don't want to have to keep the definition up to date in all those individual files, so you'll nearly always refer to an external schema, and in this case, you'll probably be familiar with this already. This is an AFCARS schema. The out-of-home care--the out-of-home care schema that defines the data in an out-of-home care file.

Now the schema, as you saw from the previous screen, that XML file was pointing to the out-of-home care schema, and so what does that do? What is the purpose of that? The purpose of that, the schema defines how the data must be built. So what if you built your XML file wrong? Well, you want to validate against the schema. Validating against the schema simply means making sure you're invoking the

schema or calling the schema or referencing the schema to do a check of all your data to see if you've made any mistakes in how you built it such as naming the tags wrong, including invalid data, et cetera. Now unfortunately, this screen is a little bit blurry. You might be able to see here down at the bottom once you've actually hooked your XML up to a schema in an editor--and elsewhere, I've given you a--I've done a short video that explains what kind of editors you can use and how you associate your XML with the schema that validates it. Here is an example of a screenshot of when you have your schema hooked up to your XML, when your XML is pointing to a schema and your editor is using the schema to validate the data, you get very legible error messages that are almost all--almost plain English, and hopefully you can see this example here. I've put in an invalid birthdate. Invalid value is clearly not a valid birthdate, and you can see it's flagged with squiggly red lines. You can hit a control button to see the explanation, and it says--it's a little bit--it's a little bit tech speak, but it says it doesn't match the pattern that's required for a date. So that is the purpose of--that is how the schema will work in your practical day-to-day work. You have to have your schema working if you're gonna be looking at XML to catch real-time errors in how you're constructing your XML.

OK. On the previous screen, I showed you how validation should look when you're working with your XML and I also described the different kinds of schemas you can have. You're gonna be using XML schema, which you might be acquainted with this. In "Technical Bulletin 21," you can download a zip file that contains this schema or the two schemas, one that describes the out-of-home care file, one that describes the adoption file. This one you can probably tell is the schema that defines how to build an out-of-home care file. Now let's kind of--hopefully, you can see this pretty clearly. Can you see this, audience?

Woman: Yes.

Brendan Martin: OK, good. Let's just start right from the top. This top is the most kind of intimidating and obscure-looking part. The top line is simply called the prolog, whatever. You just have to have that thing at the top to announce that this is an XML file, and then the stuff right underneath that, that long line with all those URLs, this is kind of conceptually difficult to grasp, but it's worth understanding. What it looks like is it

looks like this document is magically going out to these websites, and maybe it's checking your XML, maybe it's--you know, like it's doing something out there. In fact, these are not URLs. There's not the software going out to the web to somehow get definitions or something. These are what are called namespaces, and I'll give you another slide that illustrates a little bit how that works. Here's an example of the top of your out-of-home care file, and you can see the top line.

It starts with this data element, although it's prefixed by this "acf:". Now, what does that mean? What it means is that ACF is what's called a namespace prefix, and it's marking--you might have just written a data element, that is a tag called data--angle bracket data closing bracket. Why might that be a problem? Well, let's say you want to integrate your data with other agencies or you want to somehow merge data with other types of data. Data is a pretty generic way to name an element, and so there would be confusion about the elements. The namespace, the principal reason for having namespaces is to uniquely define an element. Now that that's fairly easy to grasp, so you'll see that--oops! Sorry about that. You'll see that--you'll see that we're calling th data element, which is--the top level element, is called the root element. That's the first element, and it contains all the descendant elements below it. We're defining it as an ACF type data.

So what are these, in fact, URLs after all? They're not really URLs. They're what are called URIs. It's a fine distinction, but it's not even worth explaining here right now, but it's just the name, and let's take a look at this third line, which will illustrate kind of how these namespaces work at the top of your file. I described to you a couple of slides earlier about how you want your XML to point to a schema. So what this line here, this is the only place where you're actually going out and getting some external file. This circled schema ooh.xsd, xsd being the--the--an XML schema file type. It's the file extension for XML schema. This is actually saying "Hey, your schema is in your directory with your XML file. It's not qualified with any directories, et cetera." It's actually going out to get this, and so what this line xsi schema location is saying, it's saying, "For anything in this namespace, this crazy URL, use this XML schema to validate it." So that now that might be clearer. This namespace with this long gobbledygook URL is--we're giving it a kind of a shorthand name of ACF, and ACF, our data element at the top, and all our data in our file is ACF type data. So you go down, and you use this schema to validate that data. The ACF prefix is largely for legibility. You don't want to have to type out this URL-looking thing colon before any element that is defined as part of that

namespace. It's a little bit tricky, but that's what that means. You can see there's, there's another namespace XSI. It's not--I don't think it's actually used in your XML file, but it is used elsewhere in schema files, but anyway, that's what that means. Your data element at the top is a special kind of data element that we define as ACF, and we use this schema circled in red to validate anything in that namespace.

So why do they have this URL notation that confuses you and makes you think it's going to the web? One of the reasons is is that it's kind of--it's you kind of own that--you own that URL or URI. Your business or agency or installation is associated with that, so you can easily extend it with slashes that have subdirectories, et cetera. It's a logical--it's closely--it's associated with your business agency or installation, and you can define other namespaces under other namespaces, et cetera, and it's relatively clear what the relationship is. Anyway, that's what that means. I don't know if I explained that well. It's a little bit tricky to grasp, but hopefully you'll remember it.

Let's look at some other elements in the XML schema. I earlier mentioned attributes. You can see most of the elements in this file have type attributes. Now, what is a type attribute? Type is a pretty generic attribute name, but once we've--but, uh...it's--what we've done here is you can see "acf:twoOrThreeChars". It's pretty intuitive. We're somehow saying the agency name has to be 2 or 3 characters, right? And how do we how did we do that? As you can see, it's prefixed by ACF, so what that means is we've defined a special ACF type for this element. It's a special ACF type that enforces that the data is 2 to 3 characters, and it would look like this. This is far less readable, a little bit cryptic, but it kind of shows you the idea is that we're defining a type that enforces rules for the content of the data, and I'll go into this very cryptic-looking stuff later. You can see it's using something from another namespace, that is the XML schema namespace. It's called a string, et-cetera.

Other kind of things worth noting here is other attributes in the XML schema are--this is--these minOccurs and maxOccurs, this tells you how many you must have and how many you can have of a given element. This element in question that I'm pointing to is the record element, OK? So this XML schema is defining how many records you can put into your XML file. Oh, and incidentally, I should have mentioned this first off the bat, there's a reason this XML schema looks like XML, and that reason is it is XML. It happens to be XML of the xs namespace variety. That is xs, which is a namespace. It's basically XML schema. So all these elements, these are XML type of

schema, and there's software that knows what to do with this. So this is the XML you actually use to build the definition for other XML.

OK. So in any case, we're down here at the record element. The record element in your out-of-home care file, it must occur--there must be at least one record and --minOccurs--and maxOccurs means you can have as many as you want, right? Incidentally, if you don't have minOccurs or maxOccurs, there's a default of one and one. So all these other elements you see below, can you have as many child record numbers per record? Can you have as many record numbers per record as you want? No, because the default is understood to be--is understood to be maxOccurs one. It should only occur once unless you specify otherwise. And can you leave it out? No, because the default is that the minimum number it must occur is one. So that's how that works. One last point on this screen is there's another type. I showed you earlier a custom type, where we said, hey, we want to call this a special kind of data that has to be 2 to 3 characters. Well, if it's some other kind that's more generic, could be a number of different characters, we don't really know how it's going to look. With some kind of data, we don't really--we don't enforce any special rules for what kind of data is in there. You would use a more generic type called a string, and this is you can see it's got this namespace. It's a special XML schema type of data, and some of you who've worked with software, you'll understand what a string is. It means a string of characters.

OK. Anyway, I don't expect you to absorb all that. I also don't expect you to go out and read the XML schema first thing off the bat. The schema is--if you're a software developer, you're gonna need the schema to know what the real rules for building the data are, but when you look at the data, it's pretty intuitive how it works and how it's built. One of the things that regularly comes up, though, or what has come up so far is that people won't be aware of this minOccurs and maxOccurs, so they won't really know what data, what elements you can optionally leave out if they're not relevant to the child's case. Those--the minOccurs is what you want to look at to see if you can leave an element out. OK. You're probably already familiar with the removal episode. Again, it's--this is showing you removal episode has many, many children, and among those children are complex elements that themselves have children. So living arrangements, for example, down here. Hopefully, you can see the screen. You can see there's an element that has this crazy name, E112_E146_living_arrangements, and we tried to name these in a meaningful way to indicate that you've got an element that spans all

these other elements, and what does that element look like or what does the definition of that element look like in the schema? It consists of living arrangement, or that is the child of that container element is a living arrangement or many living arrangements, and again, within these living arrangements are other complex elements with other--with descendants and children and descendants like the foster care family home, et cetera. So the--I've already shown you this schema's hierarchical. There are children with their own children. The descendants can go on and on, et cetera.

I thought I would show you actually one more other thing, though. I made a note here that--you remember those types I showed you? We defined special ACF namespace types. Special ACF types of data. Here's one that's a little more complex than the other one, where we say this element has to be--it has to be 2 or 3 characters. There has to be some country code or jurisdiction code, but we can allow null. So I just thought I'd show you how that's done just if you're curious. This one--I thought it would be worth showing you this. This gets to a very--I'm drilling down to a pretty deep technical level here, but to enforce the rules for what characters you can put as an element value, you've got this very cryptic-looking syntax, and this notation is something called regular expression, or regex, as people will shorten it to. And I'm not really sure what the term for this is. I would call it notation because it's common to many, many programing languages. It's common to operating systems. You can type--you can use this syntax to parse strings in many different contexts, and it's something most computer programs have to know, but it's kind of elegant and concise, even if it is cryptic. So this regular expression is saying, "Hey, this piece of data has to consist of characters." A dot means any kind of character, meaning a digit or a letter or any symbol, and within the curly braces, it says, "Hey, it needs to be at least 2 but can be up to 3 characters in length." And then you have a vertical bar saying, "Or if it's not 2 or 3 characters in length, it can be a character of zero length."

So that's how you would--that's how you would define that. Now...hopefully, I've shown you XML--shown you XML, how it works so that you can comprehend it and even take a look and try to read it and understand it and try to understand how a schema enforces the rules for building it. Why bother to do so? Because XML is, in fact, such a standard. It's used everywhere for exchanging data, and I thought the best example to give you of how prevalent it is as a way of defining data and packaging data. I'd show you what the x in docx means. You've probably never thought about why they call a Word

document docx instead of doc now. Anybody guess what the x means?

Woman: XML.

Brendan Martin: Yes. XML. So if you actually want to see this in practice, what you would do is you would--a Word file is, in fact, a zip file. That is a docx file is, in fact, a zip file. If you just change the name of the file and change the name of the file extension to .zip, you can then unzip it, and so what you have here--you can see on the left--I had a file called sample.docx, and when you unzip it, you get a sample--you get a sample folder, and there's a Word folder under that, and there's some other folders, and within that Word file, there's XML files. The actual data in a Word document these days is XML, and what did it use to look like? if you used the wrong program, not Microsoft Word, to open a Word document, you'd see gobbledygook because it could only be read by a Microsoft Word program. Now it's XML. In theory, you can use any kind of program you would use to read XML, and there are a lot more programs that know how to read XML than know how to read an old .doc Word document. And if you look at what the Word document actually looks like, you can see tons of namespaces, but the one that's prevalent in this little sample of content we have here is w, which basically means Word. And you can even probably guess what some of these tags mean. You've got, like, a t tag. That's text. You've got a you've got a p tag somewhere--oh, a p tag up here, a paragraph, et cetera. So that's how near universal the use of XML is. Same goes for an Excel file. That is also in Excel. It's now called xslx--or xlsx. I'm sorry.

And on that fascinating note--and hopefully I bet it's a surprising note. It was a surprise to me when I learned that. I'll close with the summary of what's so practical, practical and useful about XML? It's extensible. You can exchange files, and you can easily change, add elements to a file, add different kinds of data. You might not even have to exchange a new schema if you just add some tacked-on data that maybe the person you're exchanging the data with can ignore and not even have to update the schema. So it's very easily extensible. It's legible as we've seen. If you need to, you can see what it's--what you're looking at. And maybe above all is it's exchangeable. Once you exchange a DTD, you can then exchange your data, and you can modify your data and exchange a modified DTD. So I would call that a

virtuous cycle of sharing data across all platforms and businesses and agencies, et cetera. Hopefully, this has been illuminating. I'm going to close on that note. Thanks. Bye. Nice to meet you.